


Traveler™

Software Design Specification

Tammas Hicks, Thomas Joyce, Evan Pariser, Jordan Smith, Aliya Ware
https://github.com/ScriptkidHicks/Traveler

Table of Contents:

1. System Overview

As COVID-19 has drastically changed the world, so has it changed people's shopping habits. Scared to leave their homes and go to the store to buy things, many people have resorted to online shopping instead. This has spiked the demand for shipment, causing companies to become overwhelmed and stressed with the task of keeping up with deliveries. Fortunately, our Web-application Traveler™ can help businesses reduce the stress of making deliveries on time. All drivers must do is enter in their starting location and the destinations to which they want to go and they will receive a map with the most optimal order and route which to take to make all of their deliveries. This will not only decrease business owner stress, but increase customer satisfaction and loyalty as well.

Traveler™ is designed using google map's API to produce a map, which includes the routes to the user's destinations, and Prim's shortest path algorithm-to calculate in which order to visit the destinations. There are 4 main components to this web application: home page, login/Sign-up Page, the main page, and the results page. The user will be interfacing with all of these in order to be able to retrieve their optimal route solution. They must set up an account in order to access

our software. Once logged in, users will have the opportunity to enter their starting location followed by the destinations to which they need to go. Once it is all inputed in the Main page, they will submit it and be taken to a results page with the optimized route shown on a map. The software will be hosted on a website that can be run on all Windows, iOS, and MacOS platforms.

2. Software Architecture

2.1. Components and Their Functionality

1. The Driver View
    - Intro page
        - The person wishing to calculate their optimal route will see a page with different images of maps that will fade in and out upon dragging the mouse over them. They must click the “calculate route” button which will be located just before the welcome text.
    - Login/Sign-UP
        - Allows the driver to login to their existing account or create a new one. This will be the first page that the driver sees upon opening the website.
        - After signing up or logging in, the driver will be able to start entering the destinations to which they would like to travel
    - Main page
        - On this page, the user will be prompted to enter in their current location. They then will be asked to enter the rest of their destination points into separate boxes. The address box will have an auto complete feature to help avoid miss-typed locations. The addresses must be valid points on a map or else the program will not calculate the route.
    - Final Page
        - Map of the destinations with a blue path between them to symbolize the route. Each destination will be marked by red pins with the letters A-B(up to K). The marker A represents the starting and ending location, and each other marker will represent the destinations to travel to in alphabetical order.
2. The Programmer View
    - Database
        - Existing User Profiles
            - Public: Name of Driver
            - Private: Email Address, Password
        - Destinations to which the driver would like to go.
        - Starting destination of the driver.
    - Files
        - The algorithm which calculates the optimal route to each destination.
        - Google map’s API and autocomplete feature
        - All other supporting script files

2.2. Module Interaction

Within our website, all of the modules interactions produce the best experience for the user. The four main components are the intro page, login/sign-up page-for user privacy, the main page-for data gathering, and results page-for data displaying. The intro page is not the most important model for functionality. Its main purpose is to give users a more visually appealing experience.

The login/sign-up module will allow users to create new accounts or log into existing ones. This component interacts directly with the flask data base. When signing up, the authentication process will first check if the email, username, or password have been used, and if not, the information will be stored in the database privately. When a user would like to login, the program will allow them to enter their account by retrieving the previously stored authentication information.

The home page module will make sure that the user can start the program and properly enter in their address information. The homepage primarily interacts with the user and the database. The destination points must be gathered by the user and then processed by the files in the database in order to render a nice looking map back onto the browser. This page is the link between the user and the server.

The results page really just interacts with the database to gather the parsed data points and then render the map onto the screen for the user to see. It just displays the info which was provided by the database in a clean format.

2.3. Architectural Design Rationale

We decided that this would be the best way to break down our web application to solve the problem, while also being the easiest way for drivers to be able to interact with our system. To make this website easy to use and secure, it was important that we have a login page-so no one can steal the route information, a simple intro page that was aesthetically pleasing to look at, a page for address input, and a clear results page.

The login page is one module because logging in and signing up uses the same authentication process and can scan the stored data to see if there exists an account for the user trying to access the program. Each account is secured with a key that is double hashed. It was easiest to combine the checking and creation of a key into one module. This overall made it easier because we can just send one request to the backend to see if the account is there and if it is, they can log in and if not, they can set up a new one.

The intro page was an extra component we added in just to make the user more happy with their experience. Instead of going straight to route calculations, they get an

aesthetically pleasing intro page to interact with for however long they would like. Though it was not necessary, this adds an element of enjoyment to our website.

The main page is used for address input. We were debating combining this with the results page so the user can enter destinations into the prompt and the map could calculate the route on the same page, however for development, separating the 2 made it easier on the programmer end. It made handing information between pages using props easier than it would have been to do alternate renderings of the map. Also, separating the modules makes it easier to reduce cascading failure if something were to break.

The results page ends up consisting of a nicely rendered map with a list of the destinations in order from first stop to last stop. The user can see a photo of what the route looks like and have a simple readable list of where to go, it overall makes usability quite simple.

3. Software Modules

3.1. Module Intro Page

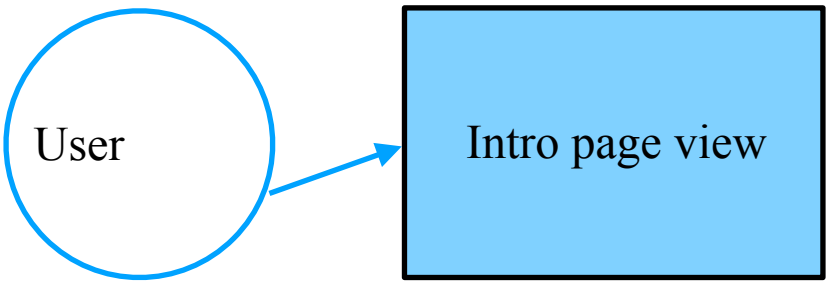
3.1.1.Primary Role and Function

This is the main page that the user sees. It's simply for visual appeal and consists of multiple images fading in and out of the screen when the mouse runs over them. There is a "Find your route solution" button which the user can click and be sent to the login/sign-up page.

3.1.2. Interface

Users can interact with the homepage in various ways. They can drag their mouse over the screen to see images pop up and see the different effects, as well as click the start button to move on the the address input page.

3.1.3. Static Model



3.1.4. Dynamic Model

Not applicable, the intro page is merely for user viewing and does not interact with any other modules

3.1.5. Design Rationale

This was created as the first page users see to produce a visually appealing intro to the program. When a user wants to use the web application they will be happier to use it if it is visually appealing to them rather than dull and boring.

3.1.6. Alternative Design

Alternatively, we were considering combining this home page and the address input page to make it easier for us as the programers and for the users. We decided however, that the user would have a more enjoyable and equally simple experience by making this page a separate introduction page.

3.2. Log In/Sign-Up

3.2.1. Primary Role and Function:

Users will have the option to sign into their existing account or create a new one. If the user tries to create a new account with the same email or password or username, they will not be allowed to and be asked to sign into their own account. The login feature works to primarily protect the user's starting destination and route to the others.

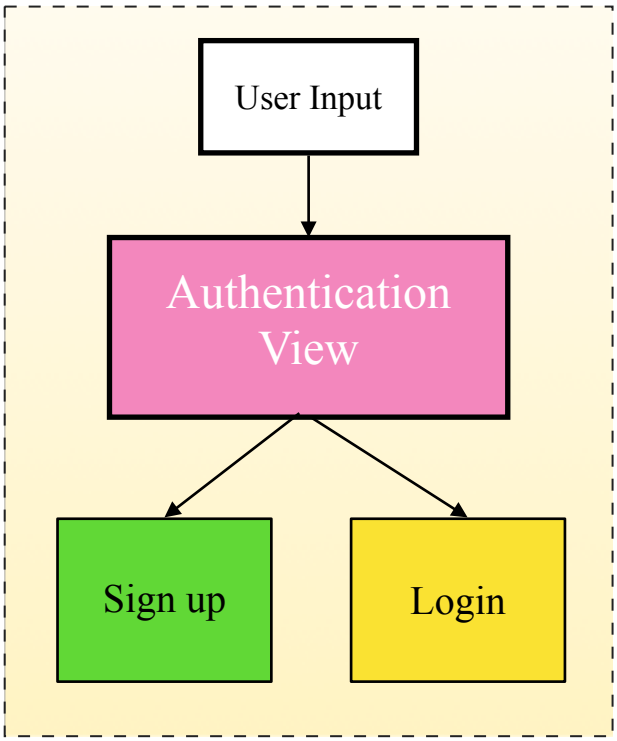
3.2.2.Interface:

The user will be immediately directed to the sign-up or log in page after the intro page. If the user does not have an account with us, they must create one, or else, they can sign in to their existing account.

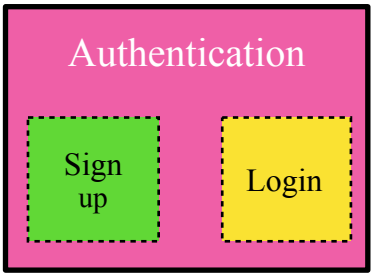
If the user needs to sign up, they will be directed to a page where they will enter in their desired user name, their email address, and whatever password they chose. Once they sign up, they will be taken to the main page where they can start entering their destinations. If the user already has an account, they will simply sign in and be taken to the main page.

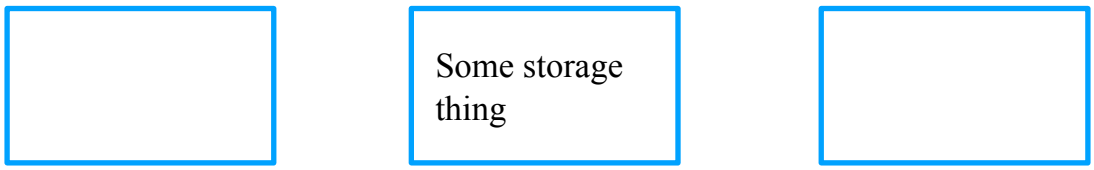
As of now, we don not have OAuth so users can create an account and be taken to the main page, or just skip it all together by entering a random username/password into the login page.

3.2.3. Static Model:



3.2.4. Dynamic Model:





3.2.5.Design Rationale:

3.2.6. Alternative Design:

3.3. Address Input Page (Main Page)

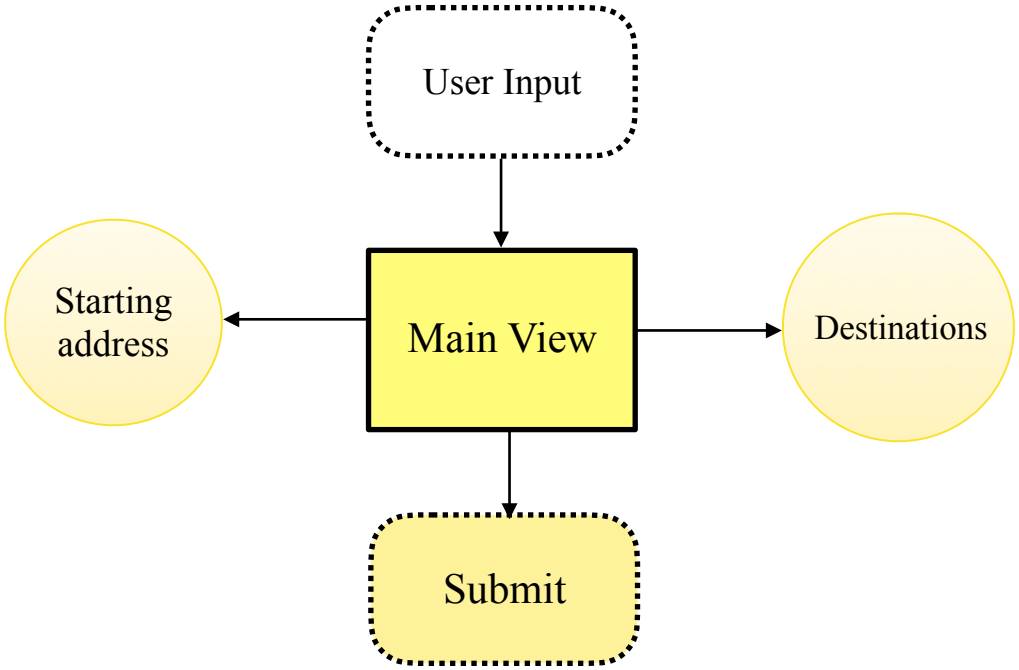
3.3.1.Primary Role and Function

The primary role and function of this page is for users to enter their addresses and submit them to get a map back. It has 10 slots in which the user can enter in the places to which they need to go and 1 slot at the top to put in their starting location. There is also an autocomplete feature that will give the user suggestions for what their address is as they type it, in order to avoid miss typed addresses

3.3.2. Interface

Users have control over their starting and end points. Each box on the screen has a light text in it to let the user know which should be used for the starting point and which should be used for destinations

3.3.3. Static Model



3.3.4. Dynamic Model

3.3.5. Design Rationale

We designed this because it is the simplest and clearest way to get the user to input their addresses. We use a similar format as most mapping sites do, as we provide boxes for the user to enter in their locations. This is the page from which the user will enter in their destinations and starting address so we wanted it to be as user friendly as possible

Also, splitting this module helped with getting a better understanding of how each component works together. The separation of this module also helped with our own understanding of the project which in turn helped our ability to develop it. The destinations are an important part of the project, so having a separate file to grab the destinations from and pass them to the backend made it overall much simpler.

3.3.6. Alternative Design:

The alternative to keeping this module on its own was to integrate it with the results page. We thought it may be nice to see the results right after you type in a destination but we ended up going a different route. Separating the results and input pages made it easier on our end for data handling and map rendering. This way, the user can clearly see a prompt of where to enter their addresses, and on the next page, a clear map rendered with a route between each destination and a list of the optimized route order next to it.

3.4. Results Page

3.4.1.Primary Role and Function

This page displays the final route map for the user to see with all of their destinations. The routes to each place will be colored blue and will be the optimal path to get to that location. The points will be labeled A-B(up to K) with A as the starting point and B(-K) as the ending point(s).

3.4.2. Interface

A visual representation of a map will be displayed for the user to see. Google Map's API key will render the points on the map with a blue line going between them as the most optimal route- calculated by our tsp algorithm

3.4.3. Static Model

3.4.4. Dynamic Model

3.4.5. Design Rationale

This was the best way we could think of to get the results outputted in a nice readable format for the user. The logic behind the route calculation is all done in a way which is unrecognizable to most users, so outputting the results in a map format makes it easy to read. The user will be able to recognize each destination and actually see how to get there rather than read how to get there.

3.4.6. Alternative Design

Interface Specification

A software interface specification describes precisely how one part of a program interacts with another. (Faulk, Young) This is not the user interface, but instead a description of services such as public methods in a class, or getters and setters. Describe the software interface that each module will make available to other software components, both internal and external. This will help to explain how components will interact with each other, what services each module will make available, how to access a module, and what needs to be implemented within a module.

Find the right abstraction for each interface specification. For example, describe the services that will be provided but not how they will be implemented within the module. An interface that reveals too much information is *over-specified* and limits freedom of implementation. (Faulk and Young, 2011)

A Static and Dynamic Model

Every software module should be described with a static or dynamic model, and probably both.

Each diagram should use a specific design *language*. There are enough languages that have been developed such that you will not likely need to devise your own. Most diagram languages emphasize a static or dynamic representation, and do not typically mix the two. For example, class diagrams are primarily static, though they hint at time-based dynamic activities in their method names. Sequence diagrams emphasize activities over time, but also list the static entities (such as the classes) that are interacting. But you do not typically see a dynamic activity indicated on an association between two classes in a class diagram.

Every diagram should be introduced with a caption that appears immediately below the diagram and should describe what is shown in that diagram. Each caption should start with “Figure <x>.” and be referenced in the body of the text as “Figure <x>.” The caption should briefly describe what the diagram shows, such as “Figure 3. A sequence diagram showing the ‘Feed a child’ use-case.”

Diagrams can be hand-drawn and scanned in. There are some advantages to hand-drawn diagrams such as they are in some ways easier to modify. But they should ideally be scanned-in rather than photographed by hand, in part to keep the file sizes down. Do not fill an SDS with large (>1MB each) high-resolution photos of hand-drawn diagrams.

Design rationale

There will be a reason that each module (and the architecture) is designed as it is. The “design rationale may take the form of commentary, made throughout the decision process and associated with collections of design elements. Design rationale may include, but is not limited to: design issues raised and addressed in response to design concerns; design options considered; trade-offs evaluated; decisions made; criteria used to guide design decisions; and arguments and justifications made to reach decisions” (See IEEE Std 1016-2009.) Your design should find a good separation of responsibility for each module. One design rationale required by the IEEE standard (1016-2009) is “a description of why the element exists, ... to provide the rationale for the creation of the element.”

Alternative designs

Your SDS for this class should include alternate designs that were considered for the architecture, and for each system or subsystem. This should result from either (a) addressing multiple alternative designs considered during the project or (b) your design evolving over the course of the project.

These alternative design ideas and diagrams can be placed in a separate section entitled “Alternative Designs” or “Earlier Designs”, or could be placed immediately after each current design, and clearly marked as an alternative or earlier design.

This would not be part of a typical SDS but is included here to reinforce the idea that design is a process.

If you do not consider alternatives, you are not doing design. If you are not recording the alternatives that you consider, you are not engaged in a good design practice.

Describe alternative architectural decisions that were considered. (If there were no alternatives, then no “design” work was done.)

4. Dynamic Models of Operational Scenarios (Use Cases)

Every major Use Case should be described with a dynamic model such as a UML sequence diagram.

5. References

This section lists the sources cited in the creation of this template document. An SRS should reference all of the sources that it draws from. If sufficient citations are provided “in line” (at the point of reference) in the document, this section may not be necessary.

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young.

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM, 15*(12), 1053-1058.

Class Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Class_diagram.

Sequence Diagram. In *Wikipedia*, n.d. https://en.wikipedia.org/wiki/Sequence_diagram.

Google, *Google*, https://developers.google.com/maps/documentation/javascript/get-api-key#creating-api-keys

6. 7. Acknowledgements

We would like to thank google for having such an extensive collection of helpful resources we could learn about and implement, it made the project simpler to think about. We would also like to thank professor Flores for being a helpful guide and “client” for us to develop and market our product to.